Algorithmes de sortie du Piège de la Zone Ennuyeuse en apprentissage par renforcement

Landry Steve Noulawe Tchamanbe* — Paulin Melatagia Yonta*,**

* Département d'informatique, Université de Yaoundé I, Cameroun
** Sorbonne Université, IRD, UMMISCO, F-93143, Bondy, France
stevenoulawe@gmail.com — paulinyonta@gmail.com

RÉSUMÉ. Les algorithmes d'apprentissage par renforcement ont réussi au fil des années à obtenir des résultats impressionnants dans divers domaines. Cependant, ces algorithmes souffrent de certaines faiblesses mises en évidence par Refael Vivanti et al. qui peuvent expliquer la régression des agents même bien entraînés dans certains environnements : la différence de variance sur les récompenses entre les zones de l'environnement. Cette différence de variance conduit à deux problèmes : le Piège de la Zone Ennuyeuse (Boring Area Trap) et le Consultant Manipulateur. Nous observons que l'algorithme Adaptive Symmetric Reward Noising (ASRN) proposé par Refael Vivanti et al. présente des limites pour des environnements ayant les caractéristiques suivantes : longues durées de jeu et environnement à zones ennuyeuses multiples. Pour pallier ces problèmes, nous proposons trois algorithmes dérivés de l'algorithme ASRN nommés Rebooted Adaptive Symmetric Reward Noising (RASRN) : Continuous ε decay RASRN, Full RASRN et Stepwise α decay RASRN. Grâce à deux séries d'expérimentations menées sur le problème du bandit à k bras, nous montrons que ces algorithmes permettent de mieux corriger le problème du piège de la zone ennuyeuse.

ABSTRACT. Reinforcement learning algorithms have succeeded over the years in achieving impressive results in a variety of fields. However, these algorithms suffer from certain weaknesses highlighted by Refael Vivanti and al. that may explain the regression of even well-trained agents in certain environments : the difference in variance on rewards between areas of the environment. This difference in variance leads to two problems : Boring Area Trap and Manipulative consultant. We note that the Adaptive Symmetric Reward Noising (ASRN) algorithm proposed by Refael Vivanti and al. has limitations for environments with the following characteristics : long game times and multiple boring area environments. To overcome these problems, we propose three algorithms derived from the ASRN algorithm called Rebooted Adaptive Symmetric Reward Noising (RASRN) : Continuous ε decay RASRN, Full RASRN and Stepwise α decay RASRN. Thanks to two series of experiments carried out on the k-armed bandit problem, we show that our algorithms can better correct the Boring Area Trap problem.

MOTS-CLÉS : Apprentissage par renforcement, Piège de la Zone Ennuyeuse, ASRN, bandit à k bras.

KEYWORDS : Reinforcement Learning, Boring Area Trap, ASRN, k-armed bandit

1. Introduction

L'intelligence artificielle est un sous-domaine de l'informatique et des mathématiques qui s'est donné pour but de doter les machines (ordinateurs) de capacités jusqu'alors réservées exclusivement aux animaux et aux humains. Pour atteindre cet objectif, plusieurs méthodes ont été développées. Pendant longtemps, une intelligence artificielle consistait en un ensemble de faits et de règles d'inférence : c'était l'époque des systèmes experts. Cependant, depuis quelques années, la tendance est à l'apprentissage automatique ou artificiel (en anglais Machine Learning). L'apprentissage automatique [2] est un sous-domaine de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d' « apprendre » à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés. L'apprentissage automatique à son tour se subdivise en plusieurs sous-domaines dont les principaux sont : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement. On parle d'apprentissage supervisé lorsque les données utilisées pour entraîner la machine sont associées à des *étiquettes* qui représentent les bonnes réponses à la tâche que la machine doit exécuter. Si par contre les données ne sont pas étiquetées et qu'il s'agit pour la machine de découvrir des motifs à l'intérieur de ces dernières, il s'agit de l'apprentissage non supervisé. Enfin lorsqu'il s'agit pour la machine d'apprendre à réaliser une séquence d'actions dans le but de maximiser une récompense totale, il s'agit d'apprentissage par renforcement.

Ce papier porte sur l'apprentissage par renforcement. Parmi les succès récents de l'apprentissage par renforcement nous pouvons citer : en 2015, Mnih et al. [6] qui ont montré que l'apprentissage par renforcement permet de créer un programme qui joue à des jeux Atari. En 2018, Hessel et al. [4] ont combiné plusieurs techniques de l'apprentissage par renforcement pour améliorer les performances de l'algorihtme Deep Q-network sur le benchmark Atari 2600. AlphaGo Zero [9] est une nouvelle technique d'apprentissage par renforcement où l'agent apprend à jouer au jeu de Go en jouant contre lui-même.

Dans la suite de ce papier, nous présenterons à la Section 2 quelques notions fondamentales de l'apprentissage par renforcement. Dans la Section 3, nous présenterons le problème du Piège de la Zone Ennuyeuse (Boring Area Trap) ainsi que celui du Consultant Manipulateur (Manipulative Consultant) et également les solutions proposées dans l'état de l'art au problème du Boring Area Trap. Dans la Section 4, nous décrirons la solution nommée Rebooted Adaptive symmetric Reward Noising (RASRN) que nous proposons à ce problème. Dans la Section 5, les résultats des expérimentations menées avec RASRN sur le problème du bandit à k bras seront présentés. La Section 6 conclura le papier.

2. Notions fondamentales de l'apprentissage par renforcement

Dans l'apprentissage par renforcement, l'on dispose d'un agent plongé dans un *environnement* caractérisé par un ensemble *d'états (s)* et un ensemble *d'actions (a)* qu'il peut effectuer. Chaque action donne lieu à une *récompense (r)*, bonus ou malus et permet d'accéder à un autre état (sauf pour des états dits terminaux). L'objectif de l'agent est de déterminer une *politique (stratégie) optimale* ; celle qui lui permettra de maximiser son gain à long terme [2]. Une politique notée π est une fonction qui à chaque état associe

une action à exécuter. Dans certains cas, l'action exécutée par l'agent peut être le fruit du hasard, notamment au début de l'apprentissage où l'agent ne connaît pas encore son environnement : c'est ce que l'on appelle *l'exploration* qui est régulé grâce à un hyper-paramètre : ε, le *taux d'exploration*. Pour atteindre une politique optimale, l'agent procède de façon itérative en bouclant sur les deux phases suivantes [1, 2] :

1) **Phase d'évaluation :** pendant laquelle l'agent évalue la politique courante au travers de sa fonction d'utilité V (respectivement Q).

2) **Phase d'amélioration :** pendant laquelle l'agent met à jour sa politique au travers de sa fonction d'utilité V (respectivement Q).

On note V^π (respectivement Q^π) la fonction d'utilité V (respectivement Q) associée à la politique π. Les fonctions d'utilité sont des fonctions qui permettent d'estimer l'espérance de gain de l'agent étant donné un état *s* ou un couple état-action *(s,a)*. Ces fonctions sont représentées sous forme de tableau comme la Q-table pour la fonction Q. La politique optimale est caractérisée par les équations suivantes :

$$V^*(s) = \max_\pi V^\pi(s), \quad \forall s \in \mathcal{S} \tag{1}$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a), \qquad \forall s \in \mathcal{S}, \quad \forall a \in \mathcal{A} \tag{2}$$

Avec $\mathcal{S}$, l'ensemble des d'états et $\mathcal{A}$, l'ensemble des actions. Pour atteindre cette politique optimale, il existe plusieurs familles de méthodes en fonction de la connaissance que l'on a de l'environnement [1, 2] :

– **Méthodes de la programmation dynamique** : dans le cas où on a une bonne connaissance de l'environnement et donc on peut utiliser les informations locales pour améliorer la politique.

$$V^\pi(s) = \mathbb{E}_\pi\big\{r_t + \gamma V^\pi(s_{t+1}) | s_t = s\big\} \tag{3}$$

où s_t et s_{t+1} sont respectivement les états de l'agent aux instants t et $t+1$ et r_t est la récompense obtenue par l'agent après avoir exécuté une action dans l'état s_t. γ est le *taux d'actualisation* qui permet de définir l'importance des récompenses futures.

– **Méthodes de Monte Carlo** : lorsque l'on ne connaît pas l'environnement, on échantillonne une zone de ce dernier par exploration, puis l'on estime les valeurs des fonctions d'utilité, et l'on se sert de ces valeurs pour faire une mise à jour de la politique.

$$V^\pi(s_t) = V^\pi(s_t) + \alpha\big[R_t - V^\pi(s_t)\big] \tag{4}$$

où R_t est la moyenne des gains sur la zone échantillonnée et α le *taux d'apprentissage*.

– **Méthodes des différences temporelles** : Il s'agit d'un compromis entre les méthodes de Monte Carlo et celles de la programmation dynamique. On commence par échantillonner une zone de l'environnement comme pour les méthodes de Monte Carlo, mais on prend également en compte la corrélation entre les états pour faire les mises à jour comme pour les méthodes de la programmation dynamique.

$$V_t^{\pi}(s_t) = V_t^{\pi}(s_t) + \alpha\big[r_t + \gamma V_t^{\pi}(s_{t+1}) - V_t^{\pi}(s_t)\big] \tag{5}$$

où $V_t^{\pi}(s_t)$ est la fonction d'utilité V associée à la politique π à l'instant t.

De ce qui précède, il va sans dire que les algorithmes les plus utilisés en apprentissage par renforcement sont de type différences temporelles. Nous pouvons citer entre autres [1, 2] : Q-learning (algorithme hors-politique, qui fait sa mise à jour sur la base de l'action qui maximise le gain de l'agent à l'état suivant), SARSA (algorithme sur-politique, qui fait sa mise à jour sur la base de la Q-table) et des algorithmes plus génériques type Deep Reinforcement Learning (algorithmes où la Q-table est approximée grâce à un réseau de neurones). Tous ces algorithmes, bien qu'ayant fait leurs preuves au fil du temps, présentent certaines faiblesses qui peuvent expliquer une chute d'efficacité des agents même bien entraînés dans certains environnements. Dans [10] Refael Vivanti et al. mettent en évidence deux problèmes des algorithmes d'apprentissage par renforcement : **Boring Area Trap** et **Manipulative Consultant**. Nous décrivons ces deux problèmes dans la section suivante.

3. Piège de la Zone Ennuyeuse et Consultant Manipulateur

Dans le contexte de l'apprentissage par renforcement, une zone ennuyeuse est une zone avec une faible variance et une faible espérance de gain sur les récompenses en comparaison aux autres zones. Une zone est considérée ici comme une action ou un couple état-action. Le **Boring Area Trap**, en français, **Piège de la Zone Ennuyeuse** est donc une conséquence de cette *différence de variance* entre les zones de l'environnement. Il s'agit plus précisément d'une situation dans laquelle, un agent se retrouve piégé, coincé dans une zone ennuyeuse ; une fois dans cette zone, due à sa faible variance, la politique (devenue ennuyeuse elle aussi) ne suggère alors que des actions qui vont conduire l'agent à y rester, d'où le terme *piège*.

Dans le cas des algorithmes de type Deep Reinforcement Learning, l'agent utilise un réseau de neurones pour prédire quelle est la meilleure action à exécuter étant donné un état. Ce réseau de neurones est donc considéré comme un consultant auprès de l'agent. On parle de **Manipulative Consultant**, en français, **Consultant Manipulateur** lorsqu'il y a conflit d'intérêt entre l'agent et son consultant. En effet, le réseau de neurones suggère des actions qui vont conduire l'agent à se retrouver piégé dans une zone ennuyeuse, mais qui le conduiront lui (le réseau de neurones) à améliorer la précision de ses prédictions.

Dans le cadre de ce papier, nous nous intéressons au problème du **Boring Area Trap**. Pour pallier ce problème, bon nombre de solutions ont été proposées dans la littérature, parmi lesquelles, *couper l'influence des récompenses sur la politique* [7, 8], *l'utilisation d'un taux d'apprentissage faible* [10], ou encore *l'utilisation de l'exploration* [10] entre autres. Refael Vivanti et al. ont proposé *Adaptive Symmetric Reward Noising (ASRN)* [10], qui consiste à ajouter du bruit aux récompenses de façon symétrique dans le temps et dans

toutes les zones de l'environnement, et ce en fonction de leur variance. Ainsi, les zones de faibles variances verront leur variance augmenter mais les espérances de gain de ces zones resteront stables. Ce principe est mis en œuvre en deux phases comme il suit :

1) **Échantillonnage de l'environnement et détermination du bruit** : pendant les premières itérations de l'algorithme Q-learning :

- L'on estime la variance de chaque zone selon la formule suivante, inspirée de l'équation de mise à jour de l'algorithme Q-learning :

$$v_t = \left|Q_{new}(s_t, a_t) - Q_{old}(s_t, a_t)\right| \tag{6}$$
$$= \left|(1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a)) - Q(s_t, a_t)\right| \tag{7}$$
$$= \left|\alpha((r_t + \gamma \max_a Q(s_{t+1}, a)) - Q(s_t, a_t))\right| \tag{8}$$

- Pendant une période T, l'on échantillonne l'environnement. Il s'agit pour l'agent d'exécuter des actions de façon un peu aléatoire (car ε le taux d'exploration est élevé). Pendant cette période, l'on récupère chaque récompense r_t puis l'on calcul la variance v_t induite selon la Formule 8 et l'on stocke le couple (v_t, r_t) dans un tableau de taille T.

- Après cette période, l'on trie le tableau contenant les couples (v_t, r_t) selon l'ordre croissant des variances. L'on créé ensuite P paquets de taille N fixée chacun tels que $p \in \{1, \ldots, P\}$ avec :

$$T = \begin{cases} P \times N & \text{si } T \equiv 0 \pmod P \\ (P-1) \times N + N' \quad \text{avec } N' < N & \text{Sinon.} \end{cases}$$

(Sans nuire à la généralité, dans la suite l'on supposera que $T \equiv 0 \pmod P$). Dans chaque paquet, l'on range N récompenses consécutives (selon l'ordre croissant des variances associées). Parallèlement, l'on créé un tableau de taille P où l'on range la plus petite variance associée à chaque paquet de récompenses. À la fin de cette opération, l'on obtient P paquets de récompenses de taille de Nchacun et pour chacun d'eux, on a une *unique* variance v_0 qui lui est associée (la plus petite des variances associées au paquet).

- Une fois cette étape terminée, l'on calcule pour chaque paquet son écart-type selon la formule suivante :

$$S_p = STD_{1 \leq j \leq N}(r_j) \tag{9}$$

- L'on calcule ensuite l'amplitude du bruit à appliquer à chaque récompense en fonction de sa variance avec la formule suivante :

$$N_p = \sqrt{S_{max}^2 - S_p^2} \qquad \text{avec} \qquad S_{max} = \max_p S_p \tag{10}$$

2) **Bruitage des récompenses** : tout au long des itérations suivantes de l'algorithme Q-learning, l'on détermine pour chaque récompense r_t, l'amplitude du bruit N_p

à appliquer selon la variance v_t induite par la mise à jour. L'on échantillonne alors une récompense bruitée r'_t suivant la distribution :

$$r'_t \sim \mathcal{N}(r_t, N_p^2). \tag{11}$$

Il est à noter que plusieurs paquets peuvent faire référencent à une même zone, il suffit pour cela que les variances v_0 qui leurs sont associées soient proches ; ainsi les paquets de faible variance feront référencent à une zone d'ennui avec une intensité plus ou moins grande en fonction de leur variance respective et les paquets de forte variance feront quant à eux référencent à une zone d'intérêt également avec une intensité qui varie en fonction de leur variance respective.

À partir d'expérimentations menées sur le problème du bandit à 2 bras, Refael Vivanti et al. [10]. montrent que l'algorithme ASRN résout le problème du Boring Area Trap. En reprenant les expérimentations menées par de Refael Vivanti et al. [10] sur de *longues durées de jeu*, l'on observe que le problème du Boring Area Trap persiste. Cette situation apparaît également dans le cas du bandit à *plusieurs bras (nombre de bras supérieur à 2)*, sur des durées de jeu moins longues que pour le cas du bandit à 2 bras. Ces deux caractéristiques à savoir : longues durées de jeu et environnement à zones ennuyeuses multiples, sont communes à plusieurs applications. telles que : la conduite autonome [3, 11] et la peinture autonome [5].

Au vu des insuffisances que nous avons relevées dans la solution ASRN de Refael Vivanti et al. [10], nous proposons dans la section suivante des modifications de cet algorithme qui corrigent le problème du Boring Area Trap pour des applications ayant les caractéristiques sus-citées.

4. Redémarrage de l'algorithme ASRN

Le principe des algorithmes que nous proposons est de **redémarrer** à intervalle de temps régulier le processus d'échantillonnage de l'environnement et de détermination du bruit à appliquer par zone. Dans le cas de l'algorithme ASRN ce processus est appliqué une seule fois, au début de l'algorithme. Au vu des contraintes posées par l'algorithme ASRN (à savoir que le taux d'exploration doit être élevé au début de l'algorithme), nous avons modifié les fonctions de variation des hyper-paramètres ε (taux d'exploration) et α (taux d'apprentissage) de l'algorithme Q-learning. Nous proposons trois versions de l'algorithme modifié appelé **rebooted ASRN (RASRN)**. Chacun de ces trois algorithmes RASRN est différent des autres par la manière dont varient les hyper-paramètres de l'algorithme Q-learning. Ces algorithmes sont :

– **Continuous ε decay RASRN** : Il s'agit d'une version de RASRN où l'algorithme ASRN est redémarré avec la valeur *courante* de ε. Contrairement à l'algorithme ASRN qui démarre avec une valeur de ε à 1 qui décroît dans le temps (l'ensemble des valeurs de ε forme une suite géométrique décroissante de premier terme 1 et de raison ε-decay), ici, seule la première exécution de la phase d'échantillonnage de l'environnement se déroule avec une valeur de ε *maximale*, les autres se déroulent avec la valeur *courante*. L'autre particularité de cette version est que l' hyper-paramètre ε ne décroît pas jusqu'à 0 mais jusqu'à une valeur minimale (0.01 dans le cas nos expérimentations). Cet algorithme n'est pas fidèle à l'algorithme ASRN car pour bien fonctionner, ce dernier a besoin d'une

valeur de ε maximale lors de la phase d'échantillonnage de l'environnement, cependant, il permet à l'agent d'exploiter sa Q-table 99% du temps environ, Q-table qui est le résultat des exécutions précédentes.

– **Full RASRN** : Contrairement à l'algorithme précédent, celui-ci consiste à faire remonter la valeur de l'hyper-paramètre ε à 1 au début de chaque itération du démarrage de l'algorithme ; ceci pour rester fidèle à l'algorithme ASRN, mais au risque de faire fi de la Q-table construite lors des exécutions précédentes, et ce, pendant les premiers instants qui suivent le redémarrage.

– **Stepwise α decay RASRN** : Dans cette version, il s'agit de trouver un compromis entre les deux précédentes, rester fidèle à l'algorithme ASRN en faisant remonter la valeur de ε à 1 après chaque redémarrage tout en ne faisant pas fi des Q-tables des itérations passées, ceci en faisant décroître la valeur de l'hyper-paramètre α au moment de chaque redémarrage. Pour baisser cette valeur, nous avons utilisé un autre hyper-paramètre α-decay.

Dans la section suivante, nous présentons les expérimentations réalisées avec ces algorithmes qui montrent bien qu'ils améliorent l'algorithme ASRN.

5. Expérimentations

Nos expérimentations ont porté sur le problème du bandit à k bras (aussi appelé problème du bandit manchot). C'est un problème mathématique qui se formule de manière imagée de la façon suivante : un utilisateur (un agent), face à des machines à sous, doit décider quelles machines jouer. Chaque machine donne une récompense moyenne que l'utilisateur ne connaît pas a priori. L'objectif est de maximiser le gain total (cumul des récompenses obtenues) de l'utilisateur [2]. Ce problème peut également être considéré comme étant celui d'un agent devant une machine unique ayant plusieurs bras et chaque bras ayant une espérance de gain inconnue par le joueur.

Nous avons conduit deux séries d'expérimentations. Le but de la première série d'expérimentations a été de mettre en évidence le fait que malgré l'algorithme ASRN, le Boring Area Trap persiste lorsque la durée du jeu augmente, mais également de montrer comment notre solution RASRN permet de pallier ce problème.

Dans la seconde série d'expérimentations, le but était de vérifier que lorsque le nombre de bras du bandit augmente, l'algorithme ASRN ne permet plus de corriger efficacement le problème du Boring Area Trap. Nous avons également montré que notre solution RASRN se comporte mieux que l'algorithme ASRN dans ces cas.

5.1. Protocole expérimental

Dans le cas de la première série d'expérimentations, nous avons opté comme Refael Vivanti et al. [10] , pour la configuration suivante : le nombre de bras du bandit est k=2 ; l'agent n'a que deux possibilités de jeu : le *bras gauche* ou le *bras droit*. Du point de vue l'algorithme Q-learning, il s'agit d'un environnement à un seul état et à deux actions *gauche* et *droit*. L'équation de mise-à-jour devient donc :

$$Q_{new}(a_t) \leftarrow (1-\alpha)Q_{old}(a_t) + \alpha(r_t + \gamma \max_a Q(a)) \tag{12}$$

La Q-table a ainsi deux entrées Ql et Qr qui sont respectivement les espérances des gains cumulés associées aux bras gauche et droit. Sachant que le gain cumulé est $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, on va initialiser ces entrées avec les valeurs optimales :

$$Ql = \sum_{t=0}^{\infty} \gamma^t \mu_l = \frac{\mu_l}{1-\gamma} \qquad \text{et} \qquad Qr = \sum_{t=0}^{\infty} \gamma^t \mu_r = \frac{\mu_r}{1-\gamma} \tag{13}$$

où μ_l, μ_r sont les moyennes des récompenses associées respectivement au bras gauche et droit. L'on initialise les bras avec des valeurs optimales pour simuler un agent qui aurait parfaitement appris une politique optimale, mais qui, due à la différence de variance entre les récompenses des deux bras va régresser alors qu'il suit scrupuleusement la politique apprise.

Pour mener à bien nos expérimentations, nous avons initialisé chaque agent avec les valeurs suivantes : $\mu_l = 0$, $\sigma_l = 0.5$, $\mu_r = 1$ et $\sigma_r = 7$. Les hyper-paramètres de l'algorithme Q-learning sont : $\alpha = 0.1$, $\gamma = 0.9$, $\varepsilon = 1$, ε-min = 0.01 (dans le cas de l'algorithme *Continuous ε decay RASRN*), ε-decay $= 0.999$, α-decay = 0.5 (dans le cas de l'algorithme *Stepwise α decay RASRN*). Comme on peut le remarquer en regardant les valeurs des ecarts-types des bras gauche et droit, il y a une différence de variance de valeur 49, ce qui va forcément induire un problème de Boring Area Trap, raison pour laquelle nous avons choisi cet exemple extrême ; il permet de mettre en évidence le problème.

Nous avons monitoré la Q-table pour déterminer à quel moment l'agent entre dans la zone ennuyeuse, le temps qu'il y met, s'il parvient à en sortir et quand cela se produit. Les algorithmes en compétition ici sont l'algorithme ASRN (celui de Refael Vivanti et al. [10]) et les versions de RASRN que nous proposons. Le nombre de redémarrages des algorithmes RASRN a été fixé à cinq. Nous avons pour chacun de ces algorithmes instancié 10 agents qui ont résolu le problème du bandit à 2 bras dans les conditions énoncées ci-dessus. Les durées de jeu que nous avons retenues pour nos expérimentations sont : 40 000, 50 000, 60 000, 400 000, 500 000, 600 000, 1 000 000. Les métriques utilisées dans le cadre de cette série d'expérimentations sont :

– **Pourcentage moyen de bons choix pour les 10 agents :** Au vu des conditions initiales de nos expérimentations, le bon choix correspond au choix du bras *droit*. Comme son nom l'indique, il s'agira de calculer le pourcentage de bons choix pour chacun des 10 agents et d'en faire la moyenne. Cette métrique a été retenue car elle nous renseigne sur la qualité du choix opéré par l'agent, et par conséquent s'il est victime d'un Boring Area Trap.

– **Gain total moyen pour les 10 agents :** Cette métrique nous a semblé importante car le but final de l'agent est de maximiser son gain total.

Dans le cadre de notre seconde série d'expérimentations, nous avons travaillé avec des environnements qui correspondent aux versions du bandit à k bras où k vaut 2, 3, 4 et 5. L'objectif de cette série d'expérimentations a été de montrer que lorsque le nombre de zones, notamment de zones ennuyeuses croît la solution ASRN de Refael Vivanti et al. [10] voit ses résultats chuter, notamment en terme de pourcentage de bons choix. Nous avons également vérifié que grâce à nos différentes versions de RASRN, nous apportons une solution à ce problème. Les variantes du bandit à k bras utilisées ici comme environnement à chaque fois, ont été initialisées de la façon suivante : lorsqu'il s'agit d'une

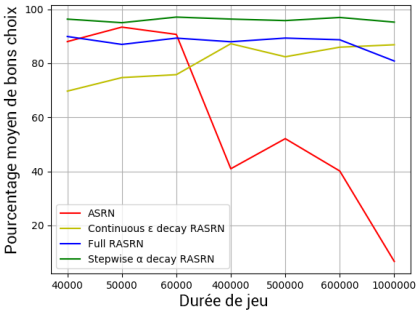

Figure 1. *Pourcentage moyen de bons choix : k=2.*

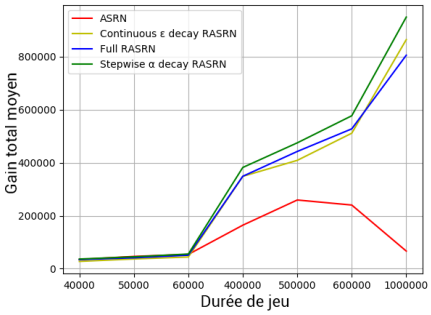

Figure 2. *Gain total moyen : k=2.*

zone ennuyeuse (b), $\mu_b = 0$, $\sigma_b = 0.5$, et lorsqu'il s'agit de la zone intéressante (i) (il y en a une à chaque fois, peu importe le nombre de bras), $\mu_i = 1$, $\sigma_i = 7$. Dans ces conditions, le choix du bras intéressant est considéré comme étant le bon, et tout autre choix est considéré comme étant mauvais, $b \in \{1, ..., k\} \setminus \{i\}$. La métrique utilisée dans ces expérimentations est *le pourcentage moyen de bons choix* effectué par les agents. Les hyper-paramètres de l'algorithme Q-learning sont les mêmes que dans la première série d'expérimentations. Les durées de jeu retenues pour cette série d'expérimentations sont : 500 000 et 600 000 épisodes.

5.2. Résultats et discussions

Dans le cadre de la première série d'expérimentations, la Figure 1, montre que lorsque la durée du jeu croît, le pourcentage moyen de bons choix effectué par les agents a une tendance décroissante lorsque ces derniers utilisent l'algorithme ASRN de Refael Vivanti et al. [10]. L'on observe également sur cette figure que lorsque les agents utilisent les algorithmes RASRN, malgré l'augmentation de la durée du jeu, le pourcentage moyen de bons choix reste assez élevé, avec une prédominance de l'algorithme Stepwise α decay RASRN sur les autres. En effet, la courbe de l'algorithme Stepwise α decay RASRN est au-dessus de toutes les autres pour les durées de jeu considérées sur la figure ; en particulier, très au-dessus de la courbe de l'algorithme ASRN lorsque la durée de jeu est supérieure ou égale à 400 000 épisodes. La Figure 2 montre que lorsque la durée du jeu augmente, le gain total moyen des agents a tendance à décroître au bout d'un certain temps (lorsque le jeu à une durée supérieure ou égale à 500 000 épisodes) pour le cas de l'algorithme ASRN ; par contre, sa tendance est toujours croissante avec les algorithmes RASRN, avec cette fois encore une prédominance de l'algorithme Stepwise α decay RASRN sur les autres. L'on note également sur cette figure que pour les durées de jeu entre 40 000 et 60 000 épisodes, les courbes des quatre algorithmes sont quasiment au même niveau, mais lorsque la durée de jeu est supérieure ou égale à 500 000 épisodes, les courbes RASRN ont une tendance croissante *exponentielle*, tandis que la courbe de ASRN à une tendance décroissante. Les Figures 3, 4, 5 et 6 quant à elles montrent l'évolution des Q-values Ql et Qr d'un agent pour les exécutions du bandit à 2 bras sur un jeu d'une durée de 1 000 000 d'épisodes. L'on observe clairement sur la Figure 3 le problème du Boring Area Trap. En effet, sur cette figure l'on peut observer à partir de l'épisode 100 000 que la courbe rouge reste au-dessus de la courbe verte, et ce de façon permanente jusqu'à la fin du jeu ; ceci traduit le fait que l'agent choisit systématiquement le bras gauche (ennuyeux)

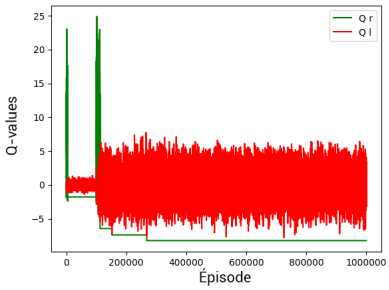

Figure 3. *Q-values : ASRN, k=2.*

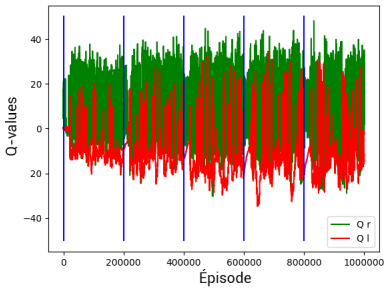

Figure 4. *Q-values : Continuous ε decay RASRN, k=2.*

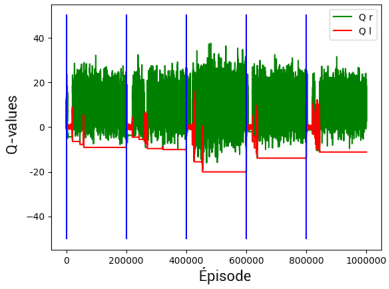

Figure 5. *Q-values : Full RASRN, k=2.*

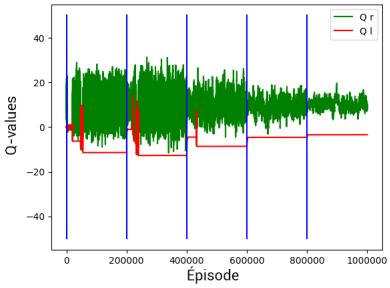

Figure 6. *Q-values : Stepwise α decay RASRN, k=2.*

au détriment du bras droit (intéressant) : il est piégé dans la zone ennuyeuse. Les Figures 4, 5 et 6 quant à elles montrent clairement une absence du phénomène du Boring Area Trap observé sur la Figure 3. En effet, sur ces figures l'on n'observe à aucun moment la courbe rouge au-dessus de la courbe verte de façon permanente ; l'on observe d'ailleurs le contraire à certains moments sur les figures 5 et 6.

Dans le cadre de notre deuxième série d'expérimentations, le Tableau 1 présente les résultats obtenus. La ligne ASRN du tableau indique clairement que le pourcentage moyen de bons choix baisse avec l'augmentation du nombre de bras du bandit, mais également avec l'augmentation de la durée du jeu. Ce qui valide expérimentalement l'hypothèse que nous avons formulée selon laquelle l'algorithme ASRN a du mal à corriger efficacement le Boring Area Trap lorsque le nombre de zones ennuyeuses de l'environnement augmente. En observant les résultats obtenus par les autres algorithmes : Continuous ε decay RASRN, Full RASRN et Stepwise α decay RASRN, le même phénomène de baisse est observé. Cependant, les algorithmes que nous proposons arrivent à maintenir à un seuil assez élevé ($> 55\%$ pour les cas illustrés dans le tableau) le pourcentage moyen de bons choix. L'on note également que l'impact de nos algorithmes RASRN dans la résolution du Boring Area Trap est d'autant plus important que la durée du jeu est grande et que le nombre de bras est grand ; comme indiqué dans le Tableau 1.

Tableau 1. *Pourcentage moyen de bons choix : k=2,3,4 et 5.*

Durées de jeu / Algorithmes	500000				600000			
	k=2	k=3	k=4	k=5	k=2	k=3	k=4	k=5
ASRN	52.09	40.95	26.43	14.29	40.12	13.08	14.46	14.50
Continuous ε decay RASRN	82.39	76.47	67.35	63.61	85.93	74.56	71.24	56.64
Full RASRN	89.31	85.26	81.58	79.18	88.67	81.83	78.73	71.54
Stepwise α decay RASRN	95.77	96.00	89.83	90.70	96.93	93.17	90.70	81.57

6. Conclusion

Le Boring Area Trap ou Piège de la Zone Ennuyeuse est un problème dont souffrent les algorithmes d'apprentissage par renforcement dans des environnements où il existe une différence de variances sur les récompenses entre les zones de ces environnements ; cette différence de variance conduit malheureusement les agents à préférer les zones de faibles variances, mais également de faibles espérances de gain. Pour pallier ce problème, Refael Vivanti et al. [10] ont proposé l'algorithme Adaptive Symmetric Reward Noising (ASRN) qui consiste à ajouter du bruit aux récompenses de façon symétrique dans le temps et dans toutes les zones de l'environnement, et ce en fonction de leurs variances. Dans ce papier, nous avons dans un premier temps montré expérimentalement que malgré cet algorithme, le problème persiste dans certaines situations (longues durées de jeu et environnement à zones ennuyeuses multiples). Dans un second temps, nous avons proposé trois variantes de l'algorithme ASRN qui améliorent le pourcentage moyen de bons choix des agents, ainsi que leur gain total moyen. Le principe de ces algorithmes nommées RASRN est le suivant : redémarrer à intervalle de temps régulier le processus de détermination du bruit à appliquer par zone. Ils sont différents les uns des autres par la manière dont sont mis-à-jour les hyper-paramètres de l'algorithme Q-learning. Il s'agit de Continuous ε decay RASRN, Full RASRN et Stepwise α decay RASRN. Grâce aux expérimentations menées sur le problème du bandit à k bras, nous avons pu mettre en évidence le fait que ces algorithmes améliorent l'algorithme ASRN. Les différences de performances entre les algorithmes RASRN et l'algorithme ASRN croissent avec l'augmentation des durées des jeux et aussi du nombre de zones ennuyeuses. En plus d'apporter une justification théorique aux résultats obtenus, nous nous proposons dans la suite de ce travail d'étendre nos algorithmes RASRN aux algorithmes SARSA et Deep Q-network, ainsi qu'à d'autres applications.

7. Bibliographie

[1] R. BELLMAN, « A Markov decision process », *Journal of Mathematics and Mechanics*, Vol. 6, No. 5 (1957), pp. 679-684.

[2] A. CORNUÉJOLS , L. MICLET, Y. KODRATOFF, « Apprentissage artificiel, Concepts et algorithmes », *Éditions Eyrolles*, 2003.

[3] A. EL SALLAB, M. ABDOU, E. PEROT, S. YOGAMANI, « End-to-end deep reinforcement learning for lane keeping assist », *arxiv.org*, arXiv preprintarXiv :1612.04340, 2016.

[4] M. HESSEL, J. MODAYIL, H. VAN HASSELT, T. SCHAUL, G. OSTROVSKI, W. DABNEY, D. HORGAN, B. PIOT, M. AZAR, D. SILVER, « Rainbow : Combining Improvements in Deep Reinforcement Learning », *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] B. JIA, C. FANG, J. BRANDT, B. KIM, D. MANOCHA., « Paintbot : Areinforcement learning approach for natural media painting », *arxiv.org*, arXiv preprint arXiv :1904.02201, 2019.

[6] V. MNIH, K. KAVUKCUOGLU, D. SILVER, A. A. RUS, « Human-level control through deep reinforcement learning », *Nature*, vol. 518, n° 7540, 2015.

[7] J. SCHULMAN, S. LEVINE, P. ABBEEL, M. JORDAN, P. MORITZ, « Trust region policy optimization », *In International Conference on Machine Learning*, (pp. 1889-1897)., 2015.

[8] J. SCHULMAN, F. WOLSKI , P. DHARIWAL, P. RADFORD, O. KLIMOV, « Proximal policy optimization algorithms », *arxiv.org*, arXiv preprint arXiv :1707.063477, 2017.

[9] D. SILVER ET AL., « Mastering the game of Go without human knowledge », *Nature*, vol. 550, n° 24270 , 2017.

[10] R. VIVANTI, T. D. SOHLBERG-BARIS, S. COHEN, O. COHEN, « Adaptive Symmetric Reward Noising for Reinforcement Learning », *arxiv.org*, arxiv :1901.10144v1, 2019.

[11] C. YOUA, JIANBO LU, D. FILEV, P. TSIOTRAS, « Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning », *Robotics and Autonomous Systems*, vol. 14, p. 1-18, April 2019.